INFORMATION AND CONTROL 13, 207–229 (1968)

Classes of Automata and Transitive Closure

NEIL D. JONES

The Pennsylvania State University, University Park, Pennsylvania 16802

A study is made of the classes of predicates accepted by three types of multitape Turing machine. In order of decreasing acceptance powers, these are the general Turing machine, the linear-bounded automaton, and the two-way multitape nonwriting automaton. Each class is shown to consist of all and only those predicates which can be defined by a corresponding class of predicate calculus formulas based on catenation, and involving as logical operators conjunction, disjunction, and a type of transitive closure on predicates of $2n$ variables.

1. INTRODUCTION

1.1 THE MAIN RESULT AND ITS CONSEQUENCES

Many studies in computational complexity have been concerned with construction of hierarchies of functions, languages or predicates. This often proceeds as follows: First, some way of representing computation is chosen—for example, the Turing machine, formal grammars, formulas of the predicate calculus or recursion schemata. Second, a series of increasingly severe restrictions is applied to the representation chosen, and it is shown that this induces a descending chain of classes of languages, functions, etc. This chain is a classification scheme, the "complexity" of a particular language, function, etc. being given by the smallest class in the chain which contains it.

The purpose of this paper is to prove the equivalence of two hierarchies of predicates. One hierarchy is given in terms of three known types of nondeterministic multitape Turing machine: the general Turing machine, linear-bounded automaton ([M], [K]) and the two-way multitape finite automaton ([KS]). The other hierarchy of predicates is based on a version of the predicate calculus, and involves formulas without quantifiers, but with transitive closure in addition to familiar logical operations.

The Turing machine types induce a chain TM $\supset$ LBA $\supset$ MTA2 of predicate classes, so that for example LBA is the class of all predicates P such that there is some linear-bounded automaton M which accepts P. A closely related hierarchy, $\mathrm{TM}_d \supset \mathrm{LBA}_d \supset \mathrm{MTA2}_d$, is induced by the corresponding deterministic series of Turing machines.

By restrictions on predicate formulas involving transitive closure we will define two more hierarchies: $\mathrm{TR}_0 \supset \mathrm{TR}_1 \supset \mathrm{TR}_2$ and $\mathrm{TR}_{0d} \supset \mathrm{TR}_{1d} \supset \mathrm{TR}_{2d}$. Our main result is the six equalities given diagrammatically:

$$\begin{array}{ccc} \mathrm{TM} & = & \mathrm{TR}_0 \\ \cup & & \cup \\ \mathrm{LBA} & = & \mathrm{TR}_1 \\ \cup & & \cup \\ \mathrm{MTA2} & = & \mathrm{TR}_2 \end{array} \quad \text{and} \quad \begin{array}{ccc} \mathrm{TM}_d & = & \mathrm{TR}_{0d} \\ \cup & & \cup \\ \mathrm{LBA}_d & = & \mathrm{TR}_{1d} \\ \cup & & \cup \\ \mathrm{MTA2}_d & = & \mathrm{TR}_{2d} \end{array}$$

This result has two main consequences. First, the characterization by means of formulas provides a new set of tools for the description and manipulation of the languages and predicates accepted by the various types of automata. This was the original motivation for this approach. Hopefully these tools will be more convenient and flexible than the explicit construction of machines.

Second, we have extended some previous theorems about automata, languages and recursive functions. Let RUD and S-RUD be the classes of rudimentary and S-rudimentary predicates, respectively (see [S]). Let CS be the class of all context-sensitive languages [C], and let ϵ_*^2 be the class of all predicates whose characteristic functions are in Grzegorczyk's function class ϵ^2 (see [R], [G]). The following have been proved:

a. Myhill: $\mathrm{RUD} \subset \mathrm{LBA}_d$

b. KS, KR: $S\text{-RUD} \subset \mathrm{MTA2}_d$

c. Ritchie: $\epsilon_*^2 = \mathrm{LBA}_d$

d. Kuroda: $\mathrm{CS} = \{P : P \in \mathrm{LBA}$, degree of P is $1\}$

As a result of Corollary 31 we see that the containments of *a.* and *b.* have been made equalities by expanding the left sides (by adding transitive closure), and that we have obtained new characterizations of ϵ_*^2 and CS.

1.2 Remarks and Preliminaries

First, some standard terminology. An *alphabet* is any nonempty finite set A. A *string* x *over* A is any sequence $x = a_1 \cdots a_m$ $(m \geqq 0)$ such that $a_i \in A$ $(1 \leqq i \leqq m)$. m is the *length* of x, written $| x |$. The string for which $m = 0$ is the *empty string*, written λ. The set of all strings over A, including λ, is written A^* and is algebraically the free semigroup with identity generated by A.

If $x = a_1 \cdots a_m$ and $y = b_1 \cdots b_n$ are strings over A, their *catenation* is the string over A given by $xy = a_1 \cdots a_m b_1 \cdots b_n$.

A *language* L over A is any subset of A^*, i.e. any set of strings over A. A natural generalization is the concept of a *predicate* P, which is any subset of the n-fold cartesian product of A^* with itself (written $(A^*)^n$). Thus a predicate is a set of n-tuples of strings. n is the *degree* of the predicate, so a language is simply a predicate of degree 1.

We will use the notation $\bar{x}_n$ to represent a sequence $x_1, \cdots, x_n$, so the n-tuple $(x_1, \cdots, x_n)$ becomes $(\bar{x}_n)$. If P is an n-ary predicate, we interpret "$(\bar{x}_n) \in P$" as "$P(\bar{x}_n)$ is true," and "$(\bar{x}_n) \in (A^*)^n - P$" as "$P(\bar{x}_n)$ is false." For example, the *catenation predicate* $C(x, y, z)$ consists of all triples of strings over A whose third component is the catenation of the first two components. We often write this simply as "$xy = z$."

Following Smullyan [S], we use the term *explicit transformation* for any operation on predicates which can be expressed in terms of: interchange of variables, identification of variables, addition of dummy variables, or the replacement of a variable by a constant (a string from A^*).

The basic automaton model used herein is a possibly nondeterministic Turing machine with a number of finite (but expandable) tapes with endmarkers, and without auxiliary symbols. An n-tuple $(x_1, \cdots, x_n)$ is accepted by an n-tape Turing machine M iff M has at least one computation which begins with $x_1, \cdots, x_n$ on its tapes and terminates in an accepting state. For convenience in constructions, we use a Wang-type machine with a program [W], rather than a state transition table. In constructions (Section 3) we will use Wang's idea of "subroutines."

It is sometimes desirable to define the acceptance of an n-ary predicate by an m-tape machine, where $m \neq n$. For example, $L = \{0^n 1^n : n \geqq 0\}$ is not acceptable by any one-tape one-way automaton, but it is quite easy to construct a two-tape one-way automaton M which accepts (x, x) iff $x \in L$. Denote by $P(x, y)$ the predicate which M accepts. Then

$L(x) \leftrightarrow P(x, x)$, i.e. the predicate $L(x)$ is obtained from $P(x, y)$ by an elementary operation on n-tuples—the identification of variables [S].

In a classification of recognitional abilities of multitape automata it seems reasonable to regard such operations as trivial. Thus we call a predicate $P(\bar{x}_n)$ *acceptable* iff P is an explicit transform of some m-ary predicate $Q(\bar{x}_m)$, and there is an m-tape automaton M which accepts all and only the m-tuples in Q.

Our concept of linear-bounded automaton differs noticeably from that of Myhill [M], in that his version has only one tape, but permits the use of auxiliary symbols. Equivalence can be shown (for alphabets with at least two letters) as follows: If M is a multitape automaton, all the tapes can be presented on a single tape by use of Myhill's "vertical packing" techniques, using a large auxiliary alphabet. The one-tape machine can then simulate the multitape machine in a straightforward manner. Conversely, any alphabet can be mapped into n-tuples of symbols from a two-letter alphabet in a very effective manner, so an n-tape machine without auxiliary symbols can simulate a one-tape machine with them.

2. DEFINITIONS AND TERMINOLOGY

2.1 Turing Machines and Other Automata

Definition 1. An n-tape *Turing machine* is a pair $Z = (A, P)$, where $A = \{a_1, \cdots, a_m\}$ is the *tape alphabet*, which does not contain the left and right end markers ϵ and ϵ' (respectively), and P is the *program*. A program is by definition a finite list of numbered instructions: $1 \cdot I_1, 2 \cdot I_2, \cdots, l \cdot I_l$, subject to the following conditions:

(a) I_l is Ac or Rj

(b) Each I_i is in one of the following forms, where $a \in A$, $b \in A \cup \{\epsilon, \epsilon'\}$, $1 \leqq t \leqq n$, $1 \leqq p \leqq l$, and $1 \leqq q \leqq l$:

0. Jp, q Jump to p or q. This instruction is used in nondeterministic machines, and different choices may be made in different computations, or even in a single computation.

1. Rt Move the head on tape t right one square, unless ϵ' is being scanned. In either case go to instruction I_{i+1}.

2. Lt Move the head on tape t left one square, unless scanning ϵ. In either case, go to instruction I_{i+1}.

3. $J(b, t)p$ Jump to instruction I_p if a "b" is being scanned on tape t, otherwise go to instruction I_{i+1}.
4. Jp Jump to instruction I_p, unconditionally.
5. Ac Stop, accepting the input.
6. Rj Stop, rejecting the input.
7. Wat Write a on the square being scanned on tape t, unless it contains ϵ or ϵ'. In either case go to instruction I_{i+1}.
8. $WXat$ Write a on the square being scanned on tape t. If it contains ϵ or ϵ', then add a new endmarker ϵ or ϵ' to the left or right end, respectively, thus extending the tape by one square. Go to I_{i+1}.

Z is *deterministic* iff P contains no instructions of Type 0; otherwise Z is *nondeterministic.* ▮

DEFINITION 2. An *instantaneous description* is any $3n + 1$-tuple $\alpha = (p, u, b_1, v_1, \cdots, u_n, b_n, v_n)$ such that:

(a) $1 \leqq p \leqq l$
(b) $b_i = \epsilon$ implies $u_i = \lambda$, for $i = 1, \cdots, n$
(c) $b_i = \epsilon'$ implies $v_i = \lambda$, for $i = 1, \cdots, n$
(d) $b_i \in A \cup \{\epsilon, \epsilon'\}$, for $i = 1, \cdots, n$. ▮

An instantaneous description represents the total state of Z at any instant. The ith tape contains $u_i b_i v_i$, including ϵ and ϵ', b_i is the symbol being scanned, and u_i and v_i are the portions of tape i to the left and right of b_i, respectively.

We shall abbreviate "instantaneous description" by "ID."

DEFINITION 3. Let α and β be ID's of Z. We say that α *yields* β *immediately*, written $\alpha \vdash \beta$, iff $\alpha = (p, u_1, b_1, v_1, \cdots, b_n, v_n)$ and β can be obtained from α by applying instruction I_p (except, of course, when $I_p = Ac$ or Rj). ▮

We omit a more formal definition of $\vdash$ since it will be completely formalized in the proof of a lemma in Section 3.

We say that α *yields* β, written $\alpha \models \beta$ iff there is a sequence $\alpha = \alpha_1, \alpha_2, \cdots, \alpha_{m-1}, \alpha_m = \beta$ of ID's such that $m > 1$ and $\alpha_i \vdash \alpha_{i+1}$ for $i = 1, 2, \cdots, m - 1$. The sequence is called a *computation.*

The ID $\alpha = (p, u_1, b_1, v_1, \cdots, u_n, b_n, v_n)$ is *accepting* iff P contains $p \cdot Ac$, *rejecting* iff p contains $p \cdot Rj$, and *final* in either case.

DEFINITION 4. If $(\bar{x}_n) \in (A^*)^n$ is an n-tuple and $Z = (A, P)$, the

initial instantaneous description of $(\bar{x}_n)$ is

$$\alpha = (1, \lambda, \epsilon, x_1\epsilon', \lambda, \epsilon, x_2\epsilon', \cdots, \lambda, \epsilon, x_n\epsilon').$$

$(\bar{x}_n)$ is *accepted* by Z iff $\alpha \models \beta$ for some accepting β, and *rejected* by Z iff $\alpha \models \beta$ some rejecting β. ▌

Intuitively, Z is given an input $(x_1, \cdots, x_n)$ on tapes 1, 2, $\cdots$, n, so it is initially scanning the left endmarkers of each tape, and is about to execute instruction 1. It operates on this configuration by a series of steps, obeying the program P, deterministically except for instructions of type 0. More specifically if α is an ID to which instruction I_p applies, then $\alpha \vdash \beta$ for exactly one β in cases 1, 2, 3, 4, 7, 8, $\alpha \vdash \beta$ for exactly two β in cases 0, and α is final (so $\alpha \vdash \beta$ for no β) in cases 5, 6.

If such a computation finally terminates the input is either accepted or rejected. In deterministic case no input is both accepted and rejected; however neither may be true, since it may never stop, in which case we say the machine *loops*. In the nondeterministic case the machine may accept, reject, or loop all in different computations on the same input.

In the following we use the symbol $\leftrightarrow$ (equivalence) ambiguously, sometimes to denote the equivalence of predicates, and sometimes to define the predicate on the left to be equal to the predicate on the right of the arrow. We write $W \rightarrow V$ (read "W implies V" to indicate that V is true whenever W is true.

DEFINITION 5. Let W and V be n-ary and m-ary predicates, respectively. We say that W is an *explicit transform* of V iff for all $x_1, \cdots, x_n \epsilon A^*$, $W(x_1, \cdots, x_n) \leftrightarrow V(\xi_1, \cdots, \xi_m)$, where each ξ_i is either one of $x_1, \cdots, x_n$, or is a constant string from A^*.

A predicate $W(\bar{x}_n)$ is *Turing-acceptable* iff there is a predicate $V(\bar{y}_m)$ and an m-tape Turing machine Z such that:

(a) W is an explicit transform of V

(b) Z accepts all $(\bar{y}_m)$ such that $V(\bar{y}_m)$ is true

(c) Z accepts no other $(\bar{y}_m)$

We say that V is *directly* accepted by Z, while W is *indirectly* accepted. W is *strongly acceptable* iff W is acceptable as above, Z is deterministic, and

(d) Z rejects all $(\bar{y}_m)$ such that $V(\bar{y}_m)$ is false. ▌

The class TM consists of all predicates W which are acceptable; TM_d consists of all predicates W which are acceptable by a deterministic Turing machine Z. ▌

DEFINITION 6. A *linear-bounded automaton* is any Turing machine without instructions of type 8 (see Definition 1); a *two-way nonwriting automaton* is one without instructions of types 7 or 8. ▮

The classes LBA and MTA2 are defined exactly as in Definition 5, with the restriction that Z be linear-bounded, or two-way non-writing, respectively. The classes LBA_d and $\mathrm{MTA2}_d$ correspond similarly to TM_d. ▮

2.2 TRANSITIVE CLOSURE AND TR_0, TR_1, TR_2

In Section 2.1 we defined basic Turing machine terminology and the predicate hierarchies induced by different types of deterministic and nondeterministic machines. We now define corresponding predicate classes TR_0, TR_1, TR_2 and TR_{0d}, TR_{1d}, TR_{2d}.

DEFINITION 7. Let $A = \{a_1, \cdots, a_m\}$

1. If $x = a^1a^2 \cdots a^k$ and $a^1, \cdots, a^k \in A$, the *length* of x is k, written $|x|$
2. If $x = a^1 \cdots a^k$ and $y = b^1 \cdots b^l$, the *catenation* of x and y is $xy = a^1 \cdots a^k b^1 \cdots b^l$
3. The catenation predicate C is given by: $C(x, y, z) \leftrightarrow xy = z$
4. xBy (x *begins* y) is true iff $y = xu$ for some $u \in A^*$
   xEy (x *ends* y) is true iff $y = ux$ for some $u \in A^*$
   xPy (x is *part* of y) is true iff $y = uxv$ for some $u, v \in A^*$ ▮

DEFINITION 8. The Logical Operators

1. If $W(\bar{x}_n)$ and $V(\bar{x}_n)$ are n-ary predicates over A, then $W(\bar{x}_n) \wedge V(\bar{x}_n)$ is by definition true iff both $W(\bar{x}_n)$ and $V(\bar{x}_n)$ are true; and $W(\bar{x}_n) \vee V(\bar{x}_n)$ is true iff either or both is true.

We use $\bigvee_{i=1}^{n} F_i$ and $\bigvee_{i=1}^{n} F_i$ as abbreviations for $F_1 \vee F_2 \vee \cdots \vee F_n$ and $F_1 \vee \cdots \wedge F_n$, respectively.

2. $\sim W(\bar{x}_n)$ is by definition true iff $W(\bar{x}_n)$ is false.

3. The *transitive closure* operator T_0 applies to a binary predicate $V(x, y)$ as follows: $T_0(V)(x, y)$ is true iff there is a sequence $x = x_1, x_2, \cdots, x_m = y$ such that $m > 1$ and $V(x_i, x_{i+1})$ is true for $i = 1, \cdots, m - 1$.

If $V(\bar{x}_n, \bar{y}_n)$ is $2n$-ary, then: $T_0(V)(\bar{x}_n, \bar{y}_n)$ is true iff there is a sequence of n-tuples

$$\bar{x}_n = \bar{x}_n^{\,1}, \bar{x}_n^{\,2}, \cdots, \bar{x}_n^{\,m} = \bar{y}_n$$

such that $m > 1$ and $V(x_n^{\,i}, x_n^{\,i+1})$ is true for $i = 1, \cdots, m - 1$.

We will call these sequences *trajectories* and we write them as: $x_1 \to x_2 \to \cdots \to x_m$, or $(\bar{x}_n^{\,1}) \to (\bar{x}_n^{\,2}) \to \cdots \to (\bar{x}_n^{\,m})$.

4. We now define the *bounded transitive closure* operators T_1 and T_2. First, define R_0, R_1 and R_2 by:

$$R_0(u, v) \leftrightarrow \lambda = \lambda$$

$$R_1(u, v) \leftrightarrow |u| \leqq |v|$$

$$R_2(u, v) \leftrightarrow uPv.$$

T_1 and T_2 are now defined by the following, where V is a $2n$-ary predicate, and $i = 1$ or 2:

(a) $T_i(V) = T_0(V)$ whenever $T_i(V)$ is defined

(b) For $i = 0$, 1 or 2, $T_i(V)$ is defined iff there is a sequence of integers $g_1, \cdots, g_n$ such that, for all $x_n, y_n \in A^*$,

$$T_0(V)(\bar{x}_n, \bar{y}_n) \rightarrow \wedge_{j=1}^{n} R_i(y_j, x_{gj}). \quad \blacksquare$$

Remarks. i. When $T_i(V)$ is defined it is transitive (on n-tuples), and is contained in any transitive predicate which contains V.

ii. For $i = 1$ or 2, $T_i(V)$ exists only if every y_j is bounded by some x_{gj}, in the sense that $R_i(y_j, x_{gj})$ is true.

iii. R_0 is used for notational consistency in the case $i = 0$.

5. Call $V(\bar{x}_n, \bar{y}_n)$ *single-valued* iff $V(\bar{x}_n, \bar{y}_n) \wedge V(\bar{x}_n, \bar{z}_n)$ implies $y_1 = z_1, \cdots, y_n = z_n$.

Then the *deterministic transitive closure operators*, T_{0d}, T_{1d}, and T_{2d} are defined and equal to T_0, T_1, and T_2 iff V is single-valued and $T_0(V)$, $T_1(V)$, and $T_2(V)$ exist.

Suppose that $B_1, \cdots, B_i$ are given *basis predicates*, and $OP_1, \cdots, OP_j$ are logical operators (e.g. conjunction, quantification, etc.). Then we denote by $[B_1, \cdots, B_i; OP_1, \cdots, OP_j]$ the class of all predicates which can be constructed from $B_1, \cdots, B_i$ by any finite number of applications of the operations $OP_1, \cdots, OP_j$. Alternately, it is the smallest class which contains $B_1, \cdots, B_i$ and is closed under $OP_1, \cdots, OP_j$. We can even view $[B_1, \cdots, B_i; OP_1, \cdots, OP_j]$ as a finitely generated partial universal algebra.

We now finish our main definitions by defining TR_i and TR_{id} for $i = 0, 1, 2$. "ET" stands for "Explicit Transformation."

DEFINITION 9. By definition, if $A = \{a_1, \cdots, a_m\}$:

$$\mathrm{MAT} = [\mathrm{C}; \wedge, \vee, \mathrm{ET}],$$

$$\mathrm{TR}_i = [C; \wedge, \vee, \mathrm{ET}, T_i] \; [i = 0, 1, 2), \text{ and}$$

$$\mathrm{TR}_{id} = [C; \wedge, \vee, \mathrm{ET}, T_{id}] \; [i = 0, 1, 2). \quad \blacksquare$$

Any predicate in MAT will be called a *matrix*, and can be constructed without any transitive closure. Note that by definition

$$\mathrm{TR}_0 \supseteq \mathrm{TR}_1 \supseteq \mathrm{TR}_2 \supseteq \mathrm{MAT}, \quad \text{and}$$

$$\mathrm{TR}_{0d} \supseteq \mathrm{TR}_{1d} \supseteq \mathrm{TR}_{2d} \supseteq \mathrm{MAT}, \quad \text{and} \quad \mathrm{TR}_i \supseteq \mathrm{TR}_{id}.$$

It is known (via Turing machine theory) that $\mathrm{TR}_0 = \mathrm{TR}_{0d}$, but the author does not know whether or not $\mathrm{TR}_i \supseteq \mathrm{TR}_{id}$ is a strict containment for $i = 1$ or 2.

3. SIMULATION OF AUTOMATA BY TRANSITIVE CLOSURE

In this section we prove that $\mathrm{TM} \subseteq \mathrm{TR}_0$, $\mathrm{TM}_d \subseteq \mathrm{TR}_{0d}$, etc., thus completing half of the characterization. We first treat the general Turing machine, then the linear-bounded and two-way nonwriting automata together.

Our technique is this: given Z, we construct a predicate $V_Z(\gamma, \delta)$ which holds iff γ and δ are successive ID's in a computation by Z, after the ID's have been encoded into the alphabet A. We show that V_Z is in the class TR_i, and that $T_i(V_Z)$ exists. It follows immediately that if W is accepted by Z, it is an explicit transform of $T_i(V_Z)$, and so that $W \in \mathrm{TR}_i$.

DEFINITION 10. Let $A = \{a_1, \cdots, a_m\}$, and $a = a_1$. If $Z = (A, P)$ is an n-tape Turing machine and $\alpha = (p, u_1, b_1, v_1, \cdots, u_n, b_n, v_n)$ is an ID of Z, then we define the encoded sequence $\boldsymbol{\alpha}$ by:

$$\boldsymbol{\alpha} = (a^p, \mathbf{u}_1, h(b_1), \mathbf{v}_1, \cdots, \mathbf{u}_n, h(b_n), \mathbf{v}_n),$$

where $\mathbf{u}_i$ and $\mathbf{v}_i$ are obtained from u_i and v_i by deleting ϵ and ϵ' (if present), and

$$h(b_i) = \begin{cases} b_i & \text{if } b_i \in A \\ aa & \text{if } b_i = \epsilon, \\ \lambda & \text{if } b_i = \epsilon' \end{cases} \qquad \text{all for } i = 1, 2, \cdots, n.$$

The $6n + 2$-ary *transition predicate* $V_Z(\gamma, \delta)$, where γ and δ represent sequences of $3n + 1$ strings is defined by:

$V_Z(\gamma, \delta)$ is true iff there is an ID α such that $\gamma = \boldsymbol{\alpha}$, and either

(1) $\delta = \boldsymbol{\beta}$ for some ID β such that $\alpha \vdash \beta$, or

(2) α is accepting and δ consists of $3n + 1$ λ's. ∎

LEMMA 11. *If Z accepts $W(\bar{x}_n)$ directly* (i.e., *without explicit trans-*

formation of its inputs), *then*

$$W(z_n) \leftrightarrow T_0(V_Z)(a, \lambda, aa, x_1, \lambda, aa, x_2, \cdots, \lambda, aa, x_n, \lambda, \lambda, \cdots, \lambda).$$

Further, V_Z *is single-valued if* Z *is deterministic.*

Proof. Immediate from the definitions of V_Z, T_0 and the initial configuration. ∎

Thus we see that to prove TM $\subseteq$ TR_0, it is sufficient to show that $V_Z \in \mathrm{TR}_0$. We now proceed to do this.

LEMMA 12. *If* $W(\bar{x}_n)$ *is a finite predicate, then* $W \in$ MAT.

Proof. Let $W = \{(\bar{b}_n^{\,1}), (\bar{b}_n^{\,2}), \cdots, (\bar{b}_n^{\,k})\}$. Then

$$W(\bar{x}_n) \leftrightarrow \bigvee_{i=1}^{k} \bigwedge_{j=1}^{n} (x_j = b_j^{\,k}).$$

For example, if $W = \{(a, b), (b, a), (b, \lambda)\}$, then $W(x_1, x_2) \leftrightarrow (x_1 = a \wedge x_2 = b) \vee (x_1 = b) \wedge (x_2 = a) \vee (x_1 = b \wedge x_2 = \lambda)$.

LEMMA 13. V_Z *is in* MAT, *for any Turing machine* $Z = (A, P)$.

Proof. Let P be $1{\cdot}I_1, 2{\cdot}I_2, \cdots, l{\cdot}I_l$. Define the predicates ID, Final and V_i as follows, where γ and δ are abbreviations for the sequences: $r, u_1, b_1, v_1, \cdots, u_n, b_n, v_n$ and $s, u_1', b_1', v_1', \cdots, u_n', b_n', v_n'$, respectively.

(a) ID (γ) is true iff $\gamma = \alpha$ for some ID α of Z

(b) Final (γ, δ) is true iff $\gamma = \alpha$ for an accepting ID α, and δ is all λ's.

(c) $V_i(\gamma, \delta)$ (for $1 \leqq i \leqq l$) is true iff there are ID's α and β such that $\gamma = \alpha$ and $\delta = \beta$, and $\alpha \vdash \beta$ by applying instruction $i{\cdot}I_i$.

It suffices to show that these are in MAT, since

$$V_Z(\gamma, \delta) \leftrightarrow \bigvee_{i=1}^{l} V_i(\gamma, \delta) \vee \mathrm{Final}\ (\gamma, \delta).$$

ID(γ) is in MAT by the following:

$$\mathrm{ID}(\gamma) \leftrightarrow r \in \{a, a^2, \cdots, a^l\} \wedge$$
$$\bigwedge_{j=1}^{n} (b_j \in A \vee (b_j = aa \wedge u_j = \lambda) \vee (b_j = \lambda \wedge v_j = \lambda)).$$

Now let $F(r)$ hold iff I_r is Ac. This is finite, and so is in MAT. Then we have: Final $(\gamma, \delta) \leftrightarrow \mathrm{ID}(\gamma) \wedge F(r) \wedge s = \lambda \wedge v_1' = \lambda \wedge b_1' = \lambda \wedge \cdots \wedge v_n' = \lambda$.

Thus ID and Final are in MAT. For the V_i we have nine cases from Definition 1. To shorten the definitions define Match_t by the following,

for $t = 0, 1, 2, \cdots, n$:

$$\text{Match}_t(\gamma, \delta) \leftrightarrow r = a^i \wedge$$
$$\wedge_{j=1}^{m(j\neq t)} (u_j' = u_j \wedge b_j' = b_j \wedge v_j' = v_j) \wedge \text{ID}(\gamma) \wedge \text{ID}(\delta)$$

Clearly Match_t is in MAT. We now define V_i, once for each case.

Case 0. I_i is Jp, q. Define

$$V_i(\gamma, \delta) \leftrightarrow \text{Match}_0(\gamma, \delta) \wedge (s = a^p \vee s = a^q).$$

Case 1. I_i is Rt. Define $R_\epsilon(\gamma, \delta)$, $R_A(\gamma, \epsilon)$ and $R_{\epsilon'}(\gamma, \delta)$ to hold iff γ yields δ by instruction Rt, and the I.D. γ is scanning either ϵ, a tape symbol from A, or ϵ', respectively. These are in MAT, by:

$$R_\epsilon(\gamma, \delta) \leftrightarrow \text{Match}_t(\gamma, \delta) \wedge s = a^{i+1} \wedge u_t = \lambda \wedge b_t = aa \wedge$$
$$u_t' = \lambda \wedge [(v_t = \lambda \wedge b_t' = \lambda \wedge v_t' = \lambda) \vee$$
$$\vee_{j=1}^{m} (v_t = a_j v_t' \wedge b_t' = a_j)],$$

and

$$R_A(\gamma, \delta) \leftrightarrow \text{Match}_t(\gamma, \delta) \wedge s = a^{i+1}$$
$$\wedge b_t \in A \wedge \vee_{j=1}^{m} (u_t' = u_t b_t \wedge v_t = a_j v_t'),$$

and

$$R_{\epsilon'}(\gamma, \delta) \leftrightarrow \text{Match}_0(\gamma, \delta) \wedge s = a^{i+1} \wedge b_t' = \lambda$$

Now V_i is in MAT, since $V_i(\gamma, \delta) \leftrightarrow R_\epsilon(\gamma, \delta) \vee R_A(\gamma, \delta) \vee R_{\epsilon'}(\gamma, \delta)$.

Case 3. I_i is J_p. Define $V_i \in$ MAT by: $V_i(\gamma, \delta) \leftrightarrow \text{Match}_0(\gamma, \delta) \wedge s = a^p$.

Cases 4, 5. Define $V_i(\gamma, \delta) \leftrightarrow a = \lambda$, so V_i is always false. Case 4 has already been taken care of by Final, and Case 5 will never be used, since we will only consider accepting computations.

Case 6. I_i is Lt. Similar to Rt.

Case 7. I_i is Wat. Define $V_i \in$ MAT by:

$$V_i(\gamma, \delta) \leftrightarrow [\text{Match}_0(\gamma, \delta) \wedge (b_t = aa \vee b_t = \lambda)]$$
$$\vee [\text{Match}_t(\gamma, \delta) \wedge b_t \in A \wedge u_t' = u_t \wedge b_t' = a \wedge v_t' = v_t]$$

Case 8. I_i is $WXat$. Define $V_i \in$ MAT by:

$$V_i(\gamma, \delta) \leftrightarrow \text{Match}_t(\gamma, \delta) \wedge b_t' = a \wedge v_t' = v_t \wedge u_t' = u_t$$

Thus, $V_i \in$ MAT in each case, so $V_Z \in$ MAT. ∎

THEOREM 14. $\mathrm{TM} \subseteq \mathrm{TR}_0$, and $\mathrm{TM}_d \subseteq \mathrm{TR}_{0d}$.

Proof. Let $U(\bar{y}_m) \in \mathrm{TM}$. By Definition 5, there is a predicate $W(\bar{x}_n)$ and an n-tape Turing machine Z which directly accepts W, such that U is an explicit transform of W. By Lemma 11 W is an explicit transform of $T_0(V_Z)$, and by Lemma 13 $V_Z \in \mathrm{TR}_0$, so $W \in \mathrm{TR}_0$. But this immediately implies that $U \in \mathrm{TR}_0$, so $\mathrm{TM} \subseteq \mathrm{TR}_0$.

If $U \in \mathrm{TM}_d$, then Z is deterministic and so V_Z is single-valued. Thus $T_0(V_Z) = T_{0d}(V_Z)$, and so $W \in \mathrm{TR}_{0d}$, which implies that $U \in \mathrm{TR}_{0d}$. Thus $\mathrm{TM}_d \subseteq \mathrm{TR}_{0d}$. ∎

3.1 LBA AND MTA2

To show that $\mathrm{LBA} \subseteq \mathrm{TR}_1$ and $\mathrm{MTA2} \subseteq \mathrm{TR}_2$, we modify the construction above to yield a new predicate V_Z^1 or V_Z^2 which behaves in the same manner as V_Z, but which satisfies the bounding conditions of Definitions 8 and 9, with the use of the predicates $R_1(x, y) \leftrightarrow |x| \leqq |y|$ and $R_2(x, y) \leftrightarrow xPy$.

LEMMA 15. *$R_1 \in \mathrm{TR}_{1d}$ and $R_2 \in \mathrm{TR}_{2d}$.*

Proof. Define V by:

$$V(x, y, u, v) \leftrightarrow [\bigvee_{i=1}^{m} (x = a_i u) \vee (x = \lambda \wedge u = \lambda)] \wedge \bigvee_{i=1}^{m} y = a_i v.$$

Then

$$|x| \leqq |y| \leftrightarrow T_{2d}(V)(x, y, \lambda, \lambda) \vee x = \lambda$$

1. For R_2, we first show that xBy and $\sim xBy$ are in TR_{2d}. Define V by: $V(x, y, u, v) \leftrightarrow \bigvee_{i=1}^{m} (x = a_i u \wedge y = a_i v) \vee (x = \lambda \wedge u = \lambda \wedge \bigvee_{i=1}^{m} y = a_i v)$ Then. $xBy \leftrightarrow T_{2d}(V)(x, y, \lambda, \lambda)$, so $xBy \in \mathrm{TR}_{2d}$.

2. For $\sim xBy$, let $x \neq_B y$ hold iff x and y differ in their first letters, or if $y = \lambda$ and $x \neq \lambda$. This is in TR_{2d} by: $x \neq_B y \leftrightarrow (x \neq \lambda \wedge y = \lambda) \vee \bigvee_{i=1}^{m} \bigvee_{j=1}^{m(i \neq j)} (a_i Bx \wedge a_j By)$.

Now, using in V above, define U by:

$$U(p, x, y, q, u, v) \leftrightarrow p = a \wedge [(V(x, y, u, v) \wedge q = a) \vee (x \neq_B y \wedge q = \lambda \wedge u = \lambda \wedge v = \lambda)]$$

Then $\sim xBy \leftrightarrow T_{2d}(U)(a, x, y, \lambda, \lambda, \lambda)$.

A sample trajectory for V is:

$$(ab, abcd) \rightarrow (b, bcd) \rightarrow (\lambda, cd) \rightarrow (\lambda, d) \rightarrow (\lambda, \lambda).$$

Two sample trajectories for U are:

$$(a, ab, abc) \rightarrow (a, b, bc) \rightarrow (a, \lambda, c) \rightarrow (a, \lambda, \lambda), \text{ and}$$

$$(a, ab, acd) \rightarrow (a, b, cd) \rightarrow (\lambda, \lambda, \lambda).$$

3. Finally, xPy is in T_{2d} as follows. Define

$$V(x, y, u, v) \leftrightarrow (xBy \wedge u = \lambda \wedge v = \lambda) \vee (\sim xBy \wedge x = u \wedge \bigvee_{i=1}^{m} y = a_i v)$$

Then $xPy \leftrightarrow T(V)(x, y, \lambda, \lambda)$. Two trajectories are:

$$(c, abcd) \to (c, bcd) \to (c, cd) \to (\lambda, \lambda), \quad \text{and}$$
$$(c, abd) \to (c, bd) \to (c, d) \to (c, \lambda). \quad \blacksquare$$

THEOREM 16. LBA $\subseteq$ TR_1, MTA $\subseteq$ TR_2, $\mathrm{LBA}_d \subseteq \mathrm{TR}_{1d}$, *and* $\mathrm{MTA2}_d \subseteq \mathrm{TR}_{2d}$.

Proof. Let Z be a linear-bounded automaton or a two-way nonwriting automaton, and let $i = 1$ or 2 accordingly. Define

$$V_Z{}^i(\bar{x}_n, p, \gamma, \bar{y}_n, q, \delta) \leftrightarrow V_Z(\gamma, \delta) \wedge \mathrm{Bound}^i,$$

where

$$\mathrm{Bound}^i \leftrightarrow \bigwedge_{j=1}^{n} [R_i(u_j', x_j) \wedge R_i(v_j', x_j) \wedge R_i(b_j', p)] \wedge$$
$$p = a_1 a_2 \cdots a_m a^5 \wedge p = q \wedge x_1 = y_1 \wedge \cdots \wedge x_n = y_n.$$

We are using γ, δ, and l as in Lemma 13. Certainly $T_i(V_Z{}^i)$ exists for $i = 1, 2$, by Definition 8. Further, $T_i(V_Z{}^i)$ equals $T_0(V_Z{}^i)$. For $i = 1$, this follows from the fact that a linear-bounded automaton cannot expand its tapes, so that at any step in its computation $|u_i| \leqq |x_i|$ and $|v_j| \leqq |x_j|$ and b_j is bounded by p. For $i = 2$, since no writing is permitted, each u_j or v_j will be part of x_j and b_j is again bounded by p.

These two facts are immediate from Lemma 15:

1. $V_Z{}^i \in T_{2d}$
2. If $W(\bar{x}_n)$ is accepted by Z, then

$$W(\bar{x}_n) \leftrightarrow T_i(V_Z{}^i)(\bar{x}_n, a, \lambda, aa, x_1, \cdots, \lambda, aa, x_n, \bar{x}_n, p, \lambda, \cdots, \lambda).$$

Where p is as before.

At this point the proof of theorem 14 applies. If $U \in$ LBA ($U \in$ MTA2), then U is an explicit transform of $T_i(V_Z{}^i)$. Since $V_Z{}^i \in T_{2d}$, it is true that $T_i(V_Z{}^i)$ and so U is in TR_i. Thus LBA $\subseteq$ TR_1 (MTA2 $\subseteq$ TR_2).

The deterministic case exactly parallels the above. $\blacksquare$

4. IMPLEMENTATION OF TRANSITIVE CLOSURE BY AUTOMATA

We now prove that TM $\supseteq$ TR_0, LBA $\supseteq$ TR_1, etc., thus completing equality, and the desired characterizations. Since each class on the

right is of the form $[C; \wedge, \vee, \mathrm{ET}, \mathrm{OP}]$, where $OP = T_i$ or T_{id}, it suffices to show two things: First, that C is in $\mathrm{MTA2}_d$ and thus in every class; and second, that each class is closed under its appropriate version (OP) of transitive closure.

LEMMA 17. *The following predicates are directly acceptable by two-way nonwriting automata:*

(1) $xy = z$
(2) $|x| \leqq |y|$
(3) xPy

Proof. Straightforward constructions of automata. In fact, one-way automata suffice for $xy = z$ and $|x| \leqq |y|$. ∎

4.1 CLOSURE UNDER $\wedge$, $\vee$ AND NEGATION

We now show that for any two Turing machines Z_1 and Z_2, of m_1 and m_2 tapes, resp., an $m_1 + m_2$-tape Turing machine Z can be constructed which in effect executes the instructions of Z_1 and Z_2 alternately, thus simultaneously simulating Z_1 on tapes 1 through m_1, and Z_2 on tapes $m_1 + 1$ through $m_1 + m_2$. This will be used to prove closure under Boolean operations.

DEFINITION 18. Let $Z_1 = (A, P_1)$ and $Z_2 = (A, P_2)$ be m_1- and m_2-tape Turing machines, respectively, and let $Z_2 + m_1$ be obtained from Z_2 by replacing all references to tape numbers t by $t + m_1$. The *sum* $Z = Z_1 + Z_2 = (A, P)$ is constructed as follows:

Let P_1 be $1 \cdot G_1, 2 \cdot G_2, \cdots, l_1 \cdot G_{l_1}$, and let P_2 (as modified) be $1 \cdot H_1, 2 \cdot H_2, \cdots, l_2 \cdot H_{l_2}$. We sketch the construction of the program P of Z, using symbolic labels (of course these can be replaced by numeric ones). P will be of the form: $L(1, 1) \cdot I(1, 1), L(1, 2) \cdot I(1, 2), \cdots, L(l_1, l_2) \cdot I(l_1, l_2)$, where the L's are labels, and for each pair G_i, H_j of instructions of P_1 and P_2, $I(i, j)$ is a sequence of instructions which will execute G_i and then H_j, and jump to the appropriate $L(k, l)$ to execute the instructions following G_i and H_j. To illustrate, we give $I(i, j)$ for two cases, as follows:

1. If neither G_i nor H_j is a "jump" instruction (Jp, Jp, q, or $J(b, t)p$), then by definition P contains the instructions:

$$L(i, j) \cdot G_i, L^1(i, j) \cdot H_j, L^2(i, j) \cdot JL(i + 1, j + 1).$$

2. If G_i is $J(b, s)p$ and H_j is $J(c, t)q$ then P contains the sequence:

$$L(i, j) \cdot J(b, s)L^3(i, j), \quad L^1(i, j) \cdot J(c, t)L(i + 1, q),$$
$$L^2(i, j) \cdot JL(i + 1, j + 1), \quad L^3(i, j) \cdot J(c, t)L(p, q),$$
$$L^4(i, j) \cdot JL(p, j + 1).$$

Case 2 is more complex because the next instruction after G_i may be either G_p or G_{i+1}, and as well H_j may be followed by H_q or H_{j+1}, so four exists are needed in the above sequence $I(i,j)$.

The remaining cases are treated similarly.

LEMMA 19. TM, TM_d, LBA, LBA_d, etc. are all closed under explicit transformation, $\wedge$ and $\vee$.

Proof. Explicit transformation is trivial by Definition 5.

Let $V_1(\bar{x}_n)$ and $V_2(\bar{x}_n)$ be accepted by m_1- and m_2-tape Turing machines (linear-bounded automata, etc.) Z_1 and Z_2. Without loss of generality we may assume that Z_1 and Z_2 have no Rj instructions, since we may replace "$i \cdot Rj$" by "$i \cdot Ji$" with no effect on the predicates accepted.

This ensures that Z_1 and Z_2 will stop if and only if they accept their inputs. For $V_1 \vee V_2$ form $Z = Z_1 + Z_2$ as in Definition 21. Then Z will stop iff either Z_1 or Z_2 stops, so Z accepts $(\bar{y}_{m1}, \bar{z}_{m2})$ iff Z_1 accepts $(\bar{y}_{m1})$ or Z_2 accepts $(\bar{z}_{m2})$ (or both). Thus by Definition 5, Z accepts $V_1 \vee V_2$.

For $V_1 \wedge V_2$ we construct a machine Z' which will stop iff Z_1 and Z_2 both stop. First obtain Z_1' and Z_2' from Z_1 and Z_2 by replacing all "$i \cdot Ac$" and "$i.R_j$" instructions by "$i \cdot Ji$." Then form $Z = Z_1' + Z_2'$ as before. This machine Z will never stop, but it will get into a "tight loop" if Z_1 and Z_2 both stop. Finally, construct Z' from Z by one modification: for each (i, j) such that G_i and H_j are both "Ac," replace "$L(i,j) \cdot I(i,j)$" in Z by "$L(i,j) \cdot Ac$." It should be clear now that Z' will accept $(\bar{y}_{m1}, \bar{z}_{m2})$ iff Z_1 accepts $(\bar{y}_{m1})$ and Z_2 accepts $(\bar{z}_{m2})$, so Z' accepts $V_1 \wedge V_2$.

This completes the proof, since if Z_1 and Z_2 are one type of automaton (linear-bounded, etc.), then $Z_1 + Z_2$ and Z' are of the same type. ∎

COROLLARY 20. *Each class* TM, TM_d, etc. *contains* MAT.

Proof. Immediate from Lemmas 17 and 19.

We now prove that the classes LBA_d and $\mathrm{MTA2}_d$ are closed under negation. A full proof is not given, since the result is known for LBA_d from Ritchie [R], and for $\mathrm{MTA2}_d$ from Kobayashi [KS].

THEOREM 21. LBA_d, $\mathrm{MTA2}_d$ *and* MTA_{1d} *are closed under negation.*

Proof. We show that acceptability is equivalent to strong acceptability in these cases. This is sufficient, since if W is strongly accepted by Z, then $\sim W$ is strongly accepted by the machine Z' obtained from Z by interchanging all occurrences of "Ac" and "Rj."

Suppose that Z is an n-tape deterministic machine which accepts $W(\bar{x}_n)$, and that Z_0 is an m-tape machine of the same type which

rejects all m-tuples $\bar{y}_m$. Form $Z_1 = Z + Z_0$. Denote by $N(\bar{x}_n)$ (and $N_0(\bar{y}_m)$) the number of steps in the computation by Z (or Z_0) which begins with $(\bar{x}_n)$ (or $(\bar{y}_m)$). Assign an arbitrary value to $N(\bar{x}_n)$ if Z loops.

Then Z_1 strongly accepts the predicate U defined by:

$$U(\bar{x}_n, \bar{y}_m) \leftrightarrow W(\bar{x}_n) \wedge N(\bar{x}_n) \leqq N_0(\bar{y}_m).$$

To show that acceptability is equivalent to strong acceptability, it is therefore sufficient to show that to each Z there corresponds a machine Z_0 of the same type and an explicit transform $(\bar{x}_n) \rightarrow (\bar{\xi}_m)$ such that $N(\bar{x}_n) \leqq N_0(\bar{\xi}_m)$ for all $(\bar{x}_n)$ accepted by Z. For this implies that $W(\bar{x}_n) \leftrightarrow U(\bar{x}_n, \bar{\xi}_m)$, and so that W is strongly accepted by Z_1.

Construction of Z_0 is straightforward. Suppose that Z has l instructions. If Z is a linear-bounded automaton, the number of distinct instantaneous descriptions in a terminating computation is bounded by $l \cdot \Pi_{i=1}^{m} [(|x_i| + 2) \cdot m^{|x_i|}]$ where m is the number of symbols in A. If we map $(\bar{x}_n)$ into $(\bar{\xi}_{2n+1}) = (\bar{x}_n, \bar{x}_n, a^l)$, it is simple by using the tapes as counters to construct a deterministic linear-bounded automaton Z_0 such that $N_0(\bar{x}_n, \bar{x}_n, a^l)$ exceeds the bound just stated.

Similarly for a nonwriting two-way automaton a bound is $l \cdot \prod_{i=1}^{n} (|x_i| + 2)$, which can be achieved again with $2n + 1$ tapes without writing. ∎

Remark. This technique is clearly extendable to a great number of computation-time-bounded automaton types. This result will be used in the proof of Theorem 23, to show that LBA_d and MTA2_d are closed under T_{1d} and T_{2d}.

4.2 Closure of TM, LBA, MTA2 under T_0, T_1, T_2

We shall prove the following two theorems:

Theorem 22. *Let V be a binary predicate such that $T_0(V)(x, y) \rightarrow R_i(y, x)$ for all $x, y \in A^*$, where $i = 0$, 1 or 2. Suppose further that V is directly accepted by a two-tape Turing machine, or linear-bounded or two-way nonwriting automaton Z, according as $i = 0$, 1, or 2. Then there is an automaton Z', of the same type as Z, which accepts $T_i(V)$.*

Theorem 23. *If V, Z and i are as above, and in addition V is single-valued and Z is deterministic, then there is a deterministic automaton Z' of the same type as Z which accepts $T_{id}(V)$.*

Theorem 22 is not quite sufficient to show that TM(LBA, MTA_2) is closed under $T_0(T_1, T_2)$, for two additional cases can occur: First,

V may have more than two variables; and second, V may be an explicit transform of a predicate U which is directly accepted by Z. However, these cases are easily handled by straightforward extensions of our methods, so for notational simplicity we shall prove only the simpler case above.

To evaluate $T_i(V)(x, y)$, Z' will generate a trajectory $x = x_1 \to x_2 \to x_3 \to \cdots$, a step at a time, comparing each new x_{j+1} with y. If y is generated in this way, the input (x, y) is accepted; if y is not found, the input is not accepted. Z' will have eight tapes, as follows:

Number	*Initial Contents*	*Use*
1.	x	Input, and work tape
2.	x	Lower bound on x_j
3.	x	Contains x_j, between bounds
4.	x	Upper bound on x_j
5.	y	For comparison with x_{j+1}
6.	x	Lower bound on x_{j+1}
7.	x	Contains x_{j+1}, between bounds
8.	x	Upper bound on x_{j+1}

It is quite possible that x_j can be shorter than the original $x = x_1$, so some special technique is needed to represent x_j on a tape which originally contains x. Since a tape cannot be shortened, we use three tapes to represent x_j : tape 3 will contain x_j, along with other symbols which make tape 3 as long as x. Tapes 2 and 4 specify, by the positions of their heads, the leftmost and rightmost (resp.) symbols of x_j. x_{j+1} is similarly represented on tapes 6, 7 and 8. Before constructing Z' explicitly, we define two auxiliary machines.

LEMMA 24. *If Z is as above, there is an automaton Z_b, of the same type as Z, which acts as follows: If Z_b is applied to the eight tapes above, it will exactly simulate the actions that Z would perform on the pair of tapes (x_j, x_{j+1}).*

Proof. Z_b is constructed from Z by modifying the program P of Z. Z_b will act on tapes 3 and 7 in exactly the same manner as Z does on its tapes 1 and 2, in as much as the actions of Z do not relate to the end markers ϵ and ϵ'. Z_b is constructed so that it will "think" the end markers ϵ and ϵ' on tapes 3 and 7 are at the positions indicated by the heads on tapes 2, 4 and 6, 8. The modifications to P are as follows:

1. Replace each instruction $k.R1$ in P by a sequence of instructions

which does the following:

(a) See if the head on tape 3 is in the same position as the head on tape 4.
(b) If not, move the head on tape 3 right.
(c) If the same, don't do anything.

2. Treat $k.L1$, $k.R2$, $k.L2$ analogously.
3. Replace $k.J(b, 1)p$ by instructions which act as follows:
   (a) See if the heads on tapes 2, 3 and 4 are in the same positions.
   (b) If all are different, perform $k.(J(b, 3)P)$ directly.
   (c) If the heads on 3 and 4 match and b is ϵ', jump to p.
   (d) If the heads on 3 and 4 match and b is not ϵ', go to the next instruction.
   (e) Analogous to (c) and (d) for ϵ and tapes 3, 2.
4. Replace $k.WXa1$ by instructions which do:
   (a) See if the heads on tapes 2, 3, 4 are in the same positions.
   (b) If all are different, perform $k.WXa3$ directly.
   (c) If 3 and 4 match, perform: $1.WXa3$, $2.WXa4$, $3.R4$, thus moving the upper bound to the right one symbol.
   (d) Similar, if 3 and 2 match.
5. Modify similarly occurrences of $J(b, 2)p$, $WXa2$, $Wa1$, $Wa2$.

Note: The step "compare the positions of heads 3 and 4" can be done as follows:

1. Move the head on tape 1 to the right end.
2. Do the following steps repeatedly, until ϵ' is found on tape 3 and/or 4:

   Move the head on tape 1 left one symbol; move the heads on tapes 3 and 4 right one symbol.

3. If ϵ' is encountered on tapes 3 and 4 simultaneously, the heads were in the same positions; otherwise they were not.
4. Restore the original head positions by repeating the following until ϵ' is scanned on tape 1:

   Move the head on tape 1 right one symbol; move the heads on tapes 3 and 4 left one symbol. ▌

LEMMA 25. *There are nondeterministic machines* RAND_0, RAND_1, *and* RAND_2 *such that:*

(a) RAND_1 *is linear bounded, and* RAND_2 *is two-way nonwriting.*
(b) RAND_i ($i = 0$, 1 or 2) *will when given the eight tapes above, generate a string* x_{j+1} *such that* $R_i(x_{j+1}, x_1)$ *is true, and then stop.*

(c) *For each* x_{j+1} *such that* $R_i(x_{j+1}, x_1)$ *is true,* RAND_i *has a computation which generates it.*

Proof. The nondeterministic jump instruction "Jp, q" is used for randomness. In RAND_2, it is clear that x_{j+1} can be obtained by moving the "bound heads" on tapes 6 and 8, since it is required that $x_{j+1}\, Px$. For RAND_1, it is necessary first to do such movement to determine the size of x_{j+1}. Since $| x_{j+1} | \leqq | x |$, no expansion is required. RAND_1 will then write a random string x_{j+1} within these bounds.

RAND_0 can be very similar to RAND_1, but with one difference: RAND_0 must be able to expand tapes 6, 7 and 8 when determining the size of x_{j+1}.

Using these guidelines, construction of RAND_0, RAND_1, and RAND_2 is straightforward. ▮

Proof of Theorem 22. Let Z_b and RAND_i be as above, for $i = 0, 1, 2$. Construct Z' to act as follows:

I. Move the head on tape 4 to the right end, so that $x_j = x$.
II. Apply RAND_i, to yield a candidate x_{j+1} on tapes 6, 7, 8.
III. Apply Z_b to evaluate $V(x_j, x_{j+1})$.
IV. If true, compare x_{j+1} and y (x_{j+1} can be found intact on tape 6). If false, stop, rejecting.
V. If $x_{j+1} = y$, accept the input.
VI. If unequal, copy x_{j+1} in place of x_j, (i.e. tape 7 to tape 3), and go to II.

The above will generate the trajectory $x = x_1 \to x_2 \to \cdots$, putting x_j on tape 3 and x_{j+1} on tape 7, for $j = 1, 2, \cdots$.

Steps I through V are either elementary or have been covered by Lemmas, and so can be done by the same type of automaton as Z. Step VI involves copying, which can be done directly by a Turing machine or a linear-bounded automaton, and indirectly by head movements for a nonwriting automaton, since x_{j+1} is a substring of x.

Suppose $T_i(V)(x, y)$ is true, so there is a trajectory $x = x_1 \to x_2 \to \cdots \to x_k = y$, such that $V(x_j, x_{j+1})$ holds for $j < k$. Then Z' will be given eight tapes initially containing (x, x, x, x, y, x, x, x) and scanned at their left ends. Step I sets $x_j = x$, and step II can generate x_2 on tapes 6, 7, 8, by Lemma 25. Z_b then accepts (x_1, x_2) and x_2 is compared to y. If unequal, x_2 is copied onto tapes 2, 3, 4. A new string x_3 is then generated on tapes 6, 7, and 8. Since $T_i(V)(x, y)$ is true, there is a correct sequence of choices of $x_2, x_3, \cdots, x_k$ by RAND_i, and so (x, y) will ultimately be accepted in at least one computation.

Conversely, if $T_i(V)(x, y)$ is false, there is no sequence of choices

leading from x_1 to $x_k = y$, so Z' does not accept (x, y) in any computation. ∎

COROLLARY 26. TM_d *is closed under* T_{0d}.

Proof. It is well known from recursive function theory that TM = TM_d. From Theorem 22, TM is closed under T_0, a fortiori closed under T_{0d}.

We now prove Theorem 23, the deterministic case. The proof of Theorem 22 cannot be used, since the construction of RAND_i necessarily involves the use of the nondeterministic "Jp, q" instructions. However, since V is single-valued, it is possible, for a given x_j, to enumerate all possible strings x_{j+1} such that $R_i(x_{j+1}, x_j)$ holds, and to compute $V(x_j, x_{j+1})$ for each until the correct value of x_{j+1} is found. Using this technique to replace step II, we construct a deterministic Z' as required.

DEFINITION 27. Let $A = \{a_1, \cdots, a_m\}$. Then

(a) if $x = a_m{}^k$ for some k then $x + 1 = a_1^{k+1}$

(b) if $x = ua_i a_m{}^k$ for some $k \geqq 0$, $i < m$ and $u \in A^*$, then $x + 1 = ua_{i+1}a_1{}^k$.

(c) if $h \geqq 2$, then $x + h = (x + h - 1) + 1$.

Let $x = b_1 \cdots b_k \in A^*$, where $b_j \in A$, $j = 1, \cdots, k$. The *Enumeration* $E_1(x)$ is the sequence $\lambda, \lambda + 1, \lambda + 2, \cdots, \lambda + h$ where h is the largest integer such that $|\lambda + h| \leqq |x|$. In fact, $h = m + m^2 + \cdots + m^{|x|}$.

The *Enumeration* $E_2(x)$ is the sequence

$$E_2 = \lambda, b_1, b_2, \cdots, b_k, b_1b_2, b_2b_3, \cdots, b_{k-1}b_k, b_1b_2b_3, \cdots, b_1 \cdots b_{k-1}, b_2 \cdots b_k, b_1 \cdots b_k.$$

REMARKS.

1. $E_1(x)$ is a list of all strings which are not longer than x. Our definition of $x + 1$ is the arithmetic sum of x and 1 in n-adic number notation.

2. $E_2(x)$ is a list of all strings which are part of x.

LEMMA 28. *There is a deterministic linear-bounded automaton* Enum_1 *and a deterministic two-way nonwriting automaton* Enum_2 *such that, for* $i = 1, 2$:

(a) When given the eight tapes as in (1) of Lemma 24, if x_{j+1} is not the last string in the list $E_i(x)$, then Enum_i will replace x_{j+1} by the next string in the list and stop, accepting.

(b) If x_{j+1} is the last string in $E_i(x)$, then, Enum_i will stop, rejecting.

Proof. Enum_1 merely has to add 1 to x_{j+1} if possible within its bounds, and it is well known that linear-bounded automata can add. Enum_2 can do its Enumeration by moving its "bound heads" on tapes 6 and 8, so construction is again straightforward. ▮

Proof of Theorem 23. We only need consider $i = 1$ or 2, since Corollary 26 handles TM_d. By Theorem 21 we may assume that Z *strongly* accepts $V(x, y)$. Let Z_b be as in Lemma 24, and Enum_i as in Lemma 28. Construct Z' to accept $T_{id}(V)$, as follows:

I. Move the head on tape 4 to the right end, so that $x_j = x$.
II. Move the head on tape 8 to the left end, so that $x_{j+1} = \lambda$.
III. (a) Apply Z_b to evaluate $V(x_j, x_{j+1})$.
(b) If $V(x_j, x_{j+1})$ is true, go to step IV.
(c) If false, copy the original x_{j+1} from tape 6 to tape 7 (it may have been destroyed by Z_b).
(d) Apply Enum_i to x_{j+1}, to obtain the next trial value in the list $E_i(x_j)$, if any.
(e) Go to III (a) if there is a next x_{j+1} in $E_i(x_j)$.
(f) Reject if the present x_{j+1} is the last string in $E_i(x_j)$.
IV. Compare x_{j+1} to y (x_{j+1} can be found on tape 6).
V. If equal, accept the input.
VI. If not equal, copy x_{j+1} onto x_j, and go to II.

By the Lemmas, each step can be performed by the required deterministic type of automaton. The computations for Z' are very similar to those of Theorem 22. The only essential difference is in steps II and III, where Z' of Theorem 22 picks an x_{j+1} at random, our present Z' enumerates all the possible values of x_{j+1} (for a given x_j), trying each one until it finds the one (if any) for which $V(x_j, x_{j+1})$ is true.

We see as before that Z' accepts $T_{id}(V)$. ▮

THEOREM 29. $\text{TM} = \text{TR}_0$, $\text{LBA} = \text{TR}_1$, and $\text{MTA2} = \text{TR}_2$; $\text{TM}_d = \text{TR}_{0d}$, $\text{LBA}_d = \text{TR}_{1d}$, and $\text{MTA2}_d = \text{TR}_{2d}$.

Proof. By Theorems 14 and 16, $\text{TM} \subseteq \text{TR}_0$, $\text{LBA} \subseteq \text{TR}_1$, $\text{MTA} \subseteq \text{TR}_2$, $\text{TM}_d \subseteq \text{TR}_0$, $\text{LBA}_d \subseteq \text{TR}_{id}$, and $\text{MTA2}_d \subseteq \text{TR}_{2d}$. By definition of the TR_i classes, Lemma 17 and 19, and Theorems 22 and 23, the reverse containments also hold. ▮

The following theorem shows that these classes are also closed under appropriate quantifiers. Proof is omitted, as the same results are proven in [*S*], [*M*] and [*KS*]. Independent and much simpler proofs can easily be constructed using Theorem 29 and transitive closure.

THEOREM 30. *Define the bounded quantifier operations* $\forall_<$, $\exists_<$, $\forall_P$

and $\exists_P$ *as follows*:

$$(\forall z)_{<y}R(\bar{x}_n, y) \leftrightarrow \forall z(\,|\,z\,|\,\leqq\,|\,y \rightarrow R(\bar{x}_n, y))$$

$$(\exists z)_{<y}R(\bar{x}_n, y) \leftrightarrow \exists z(\,|\,z\,|\,\leqq\,|\,y \wedge R(\bar{x}_n, y))$$

$$(\forall z)_{Py}(\bar{x}_n, y) \leftrightarrow \forall z(zPy \rightarrow R(\bar{x}_n, y))$$

$$(\exists z)_{Py}R(\bar{x}_n, y) \leftrightarrow \exists z(zPy \wedge R(\bar{x}_n, y)).$$

then

(i) TR_0 *is closed under* $\exists$.

(ii) TR_1 *and* TR_{1d} *are closed under* $\forall_<$ *and* $\exists_<$.

(iii) TR_2 *and* TR_{2d} *are closed under* $\forall_P$ *and* $\exists_P$.

COROLLARY 31.

(i) TR_{1d} *contains the rudimentary predicates.*

(ii) TR_{2d} *contains the S-rudimentary predicates.*

Results (i) and (ii) are the main theorems of Myhill [M] and Kreider and Ritchie [KR], respectively.

5. CONCLUSIONS AND FURTHER POSSIBILITIES

The goal of this paper has been to develop a set of tools which faithfully describe automata, and which are more convenient, comprehensible, and flexible than the more customary transition tables and programs. The success of this approach will be measured in terms of its utility and the insights which it gives into the nature and powers of automata of various types.

It is clear that this approach can be extended to cover other types of multitape automata, for example by adding different bounding relations $R_i(x, y)$. The author strongly suspects that Ritchie's classes F_i [R] can be completely described by appropriate bounding conditions R. In any case, the methods used in Lemma 19 and Theorems 21, 22 and 23 appear to be quite generally extendable to various types of two-way automata.

As a final remark, it should be noted that it is quite simple to impose conditions on $V(x, y)$ to make it correspond to the derivation rules of formal grammars, so that formal languages could be studied by these tools.

RECEIVED: February 27, 1968; revised July 26, 1968.

REFERENCES

B. BÜCHI, J. R. (1960), Weak second-order arithmetics and finite automata design. *Zeitschr. f. Math. Logik und Grundl. d. Math.* **B6**, S66–92.

C. CHOMSKY, N. (1959), On certain formal properties of grammars. *Inform. Control.* **2,** 137–167.

J. JONES, N. (1967), Formal languages and rudimentary attributes. Ph.D. Thesis, Univ. West. Ont., London, Ontario.

K. KURODA, S. Y. (1964), Classes of languages and linear-bounded automata. *Inform. Control.* **6,** 131–136.

KR. KREIDER, D. L., AND RITCHIE, R. W. (1966), A basis theorem for a class of two-way automata. *Zeitschr. f. Math. Logik und Grundl. d. Math.* **12,** 243–255.

KS. KOBAYASHI, K., AND SEKIGUCHI, S. (1966), On the class of predicates definable by two-way multitape finite automata, *J. ACM* **13,** 236–261.

M. MYHILL, J. (1960), Linear-bounded automata, *Wright Air Dev. Div. Tech. Note* 60–165.

R. RITCHIE, R. W. (1963), Classes of predictably computable functions, *Trans. Am. Math. Soc.*, **106,** 1 139–173.

Ro. ROSENBERG, A. (1965), Nonwriting extensions of finite automata. Report BL-39, Harvard University.

S. SMULLYAN, R. M., "Theory of Formal Systems," Princeton, New Jersey 1961.

T. TURING, A. M. (1963), On computable numbers with an application to the entscheidungs-problem, *Proc. London Math. Soc.*, Sec. 2, **42,** 230–265.

W. WANG, H., "A Survey of Mathematical Logic," Ch. VI, North-Holland Publ. Co., Amsterdam, 1963.